

Adversarial Training with Ladder Networks

Juan Maroñas Molano
 jmaronasm@gmail.com¹, Alberto Albiol Colomer
 alalbiol@iteam.upv.es², and Roberto Paredes Palacios
 rparedes@dsic.upv.es³

¹Departamento de Comunicaciones, Universidad Autónoma de Madrid

²Departamento de Comunicaciones, Universitat Politècnica de València

³Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València

November 9, 2016

Abstract

The use of unsupervised data in addition to supervised data has lead to a significant improvement when training discriminative neural networks. However, the best results were achieved with a training process that is divided in two parts: first an unsupervised pre-training step is done for initializing the weights of the network and after these weights are refined with the use of supervised data. Recently, a new neural network topology called Ladder Network, where the key idea is based in some properties of hierarchichal latent variable models, has been proposed as a technique to train a neural network using supervised and unsupervised data at the same time with what is called semi-supervised learning. This technique has reached state of the art classification. On the other hand adversarial noise has improved the results of classical supervised learning. In this work we add adversarial noise to the ladder network and get state of the art classification, with several important conclusions on how adversarial noise can help in addition with new possible lines of investigation. We also propose an alternative to add adversarial noise to unsupervised data.

1 Introduction

Learning unconditional distributions $p(x)$ using unsupervised data can be useful for learning a neural network that models a conditional distribution $p(t|x)$ where t represents the target of the task. Learning such distributions can be

used to initialize the weights of the networks, using the supervised data to refine the parameters and adjust them to the task at hand. This pre-trained neural network has already learnt important features to represent the underlying distribution of the data x .

Classical approaches for learning $p(x)$ and use them for then learning $p(t|x)$ are under two subsets. Deep belief networks [Hinton and Salakhutdinov, 2006] are based on training pairs of Restricted Boltzmann Machines (RBM), which are a kind of probabilistic energy-based model (EBM) [Lecun et al., 2006][Lecun et al., 2005], and then perform a finne-tunning of the parameters. Deep boltzmann machines [Salakhutdinov and Hinton, 2009] are EBM with more than one hidden layer to create a deep topology to after perform a fine tuning. On the other hand autoencoders [Hinton, 1990] are neural networks where the output target is the input. The autoencoder is divided in two parts: encoder and decoder. The encoder takes the input x and start reducing the dimensionality where each hidden layer h of the neural network represents a dimension. The decoder has the same topology of the encoder but starts from the last layer of the encoder (is shared between both parts) and perform operations to have an output \bar{x} which should be as closed as possible to the input. The autoencoder is trained to achieve this property by minimizing the sum of squared error between the input and the reconstruction. We then take the pretrained encoder and perform a finne-tunning of the parameters using supervised data.

2 Semi Supervised Learning and Ladder Networks

Semi supervised learning implies learning $p(x)$ and $p(t|x)$ at the same time. This means using supervised and unsupervised data in the same learning procedure. The key idea of semi supervised learning is that unsupervised learning should find new features that correlates well with the already found features suitable for the task. This suitability is driven by the supervised learning procedure.

Mixing this learning schemes can end up stalling the learning procedure for the fact that the targets of the learning schemes are different. On one side unsupervised learning tries to encode all the necessary information for reconstructing the input. On the other supervised learning is more focused on finding abstract and invariant features (at different levels of invariability) for discriminate the different inputs. Unsupervised features such as relative position or size in a face description maybe not necessary for a discriminative task and discriminative features typically do not have information about data structure so are useless to represent x .

To perform semi supervised learning the key idea is that unsupervised learning

should be able to discard information necessary for the reconstruction, and encode this information in other layer. For example suppose supervised learning finds useful to have a characteristic in a layer h_l where l represent a particular layer. At this level unsupervised learning needs some kind of representation to keep the reconstruction error low and this information is useless for supervised learning performance. If unsupervised learning could be able of representing this information in another level l' we could still keep the reconstruction error low and supervised learning could have the information needed at this level.

2.1 Latent Variable Models

Latent variable models are models for learning unconditional probability distributions with the particularity that given a latent variable we can reconstruct the observed variable by means of a likelihood probability distribution. This means the procedure not only depends on h but also on a random procedure so in some way the model adds their own bits of information for the reconstruction. More formally for a discrete distribution:

$$p(x) = \sum_{\forall h} p(x|h) \cdot p(h) \quad (1)$$

where $p(x|h)$ can be modelled like:

$$x = f_{\theta}(h) + n_{\theta_n} \quad (2)$$

that is the mean of the likelihood distribution is given by the projection of h to the observed space and the deviation is given by some noise process. We can learn the parameters using EM [Dempster et al., 1977]. The main bottleneck of EM in some latent variable models (like RBM) is that implies computing the posterior probability $p(h|x)$ of the latent variable, which is sometimes mathematically intractable.

The structure of this models fits well with the semi supervised learning paradigm because it allows discarding information for the fact that this information can be somehow added to the reconstruction. However a one hidden layer latent variable model is unable to discard information because it needs to represent all the necessary information to represent $p(x)$ in the hidden layer. The solution is to use hierarchical latent variable models where the information can be somehow distributed between the different variables and each variable is able of adding their own bits of information.

The two key ideas of this models are that modelling the observed variable as a probability distribution implies that a hidden variable can somehow add information (so we can discard information that is then added) and that making a hierarchy allows information discarding.

2.2 Ladder Networks

Ladder Network [Valpola, 2015] is neural network topology that implements the key idea of latent variable models that make suitable mixing supervised and unsupervised learning at the same time. The topology of the network is given by figure 1:

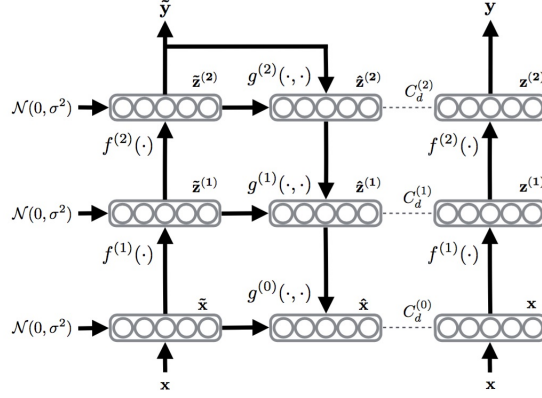


Figure 1: Ladder Network Topology [Rasmus et al., 2015]

where as we can see is a kind of autoencoder with lateral connections between the encoder and the decoder, at each level of the network. The decoder reconstruct a variable in level l using the above reconstructed variable \hat{h}_{l+1} and the corrupted variable at the same level \tilde{h}_l using a denoising function $g(\cdot, \cdot)$. This topology allows information discarding at any level, as long as it is needed. The reconstruction not only depends on the level above (and so depends on all the preceding levels) but also on the encoder part. This means their can be extra added information to the reconstruction (as the latent variable models did) and we can loose information at any level that is encoded in other levels. Note that with this topology any lower level can be influence by any signal in any higher level no matter if it is from the encoder or decoder path. This means that the features the neural network learns are somehow distributed through the network as long as the supervised learning refines which kind of feature need at any level.

The minimized cost is given by the expression:

$$C = C_s + C_u \quad (3)$$

where subindex represent supervised and unsupervised. The supervised cost is the cross entropy, that is (for only one sample X):

$$C_s = -\frac{1}{K} \sum_{k=1}^K \log P(\tilde{Y}_k = \hat{Y}_k | \tilde{X}) \quad (4)$$

where K is the layer dimension and \hat{Y} is the target. We use the corrupted output as the target to regularize. The unsupervised cost is the weighted sum of a reconstruction measure at each level:

$$C_u = \omega_0 \cdot (\|x - \bar{x}\|_2)^2 + \sum_{l=1}^L \omega_d \cdot (\|z_l - \bar{z}_l\|_2)^2 \quad (5)$$

This unsupervised cost allows deep architectures because each parameter of the topology can be well trained and is difficult to find the gradient vanishing problem.

2.2.1 Denoising Principle

We should pay special attention to the added noise in the encoder. This noise serves for two purposes. The first one is implementing the denoising principle of the denoising autoencoder [Vincent et al., 2008][Vincent et al., 2010] which serves as a good regularizer.

Noise is also added to force the $g(\cdot, \cdot)$ function to use the information in the above layer to reconstruct the signal because the signal which minimizes the cost at a level l is just h_l . This avoid the reconstruction function just copy the encoded signal to the decoder. One more special thing to remark is that the ladder network uses batch normalization for two purposes. First is avoid internal covariate shift [Ioffe and Szegedy, 2015] and the other is because we have to avoid the encoder just output constant values, as these are the easiest ones to denoise [Rasmus et al., 2015].

3 Adversarial Noise

Suppose we take a training data X from our distribution. X is an 8-bit normalized image which means there is a precision error of $\frac{1}{255}$. Now we convert this image to floating point and add a perturbation $\tau < \frac{1}{2 \cdot 255}$. If we then convert this new corrupted sample $\tilde{X} = X + \tau$ to an 8-bit image it is clear that $X = \tilde{X}$ because the perturbation lies in the same quantification interval. Our model should correctly classify this sample. [Szegedy et al., 2014] discover that neural networks are not robust to adversarial examples, that is, examples nearly similar but that highly increase the misclassification error.

The first thing we did is see if the ladder network was robust to adversarial examples. For that reason we trained one model which had a 0.51% error on the test set. We then corrupt the test set with adversarial noise and with random noise ensuring the power from both noises was the same.

Adversarial examples were computed, following [Goodfellow et al., 2014], like:

$$\tilde{X}_a = X + \tau \cdot \text{sign}\left\{\frac{\partial C_s}{\partial x}\bigg|_X\right\} \quad (6)$$

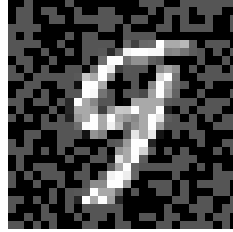
where τ ensures the perturbation is under the quantification error. Note that the energy of the adversarial noise is $\tau \cdot K$ where K is the input dimension and we can corrupt a sample using random gaussian noise like:

$$\tilde{X}_r = X + \tau \cdot \text{sign}\{q\} \quad (7)$$

where q is a samples of dimensionality K drawn from a zero mean gaussian distribution $\mathcal{N}(0, I)$. The power of these noises is the same and is K . The next figure shows a sample of the MNIST test set corrupted with both types of noise. As we see we cannot say which sample will suppose a higher missclassification error. But we can see the differences in the test error. We conclude the ladder network is not robust to adversarial examples.



(a) X_a . Error percentage 6.87%



(b) X_r . Error percentage 0.61%

3.1 Unsupervised Data

It is clear that the adversarial noise is focus on meaasuring how robust a discriminative network is. This is the reason for computing adversarial examples using the derivative of the supervised cost. Adversarial noise is then applied to purely supervised learning.

However we can easily add adversarial noise to unsupervised data taking the $\text{argmax}\{\cdot\}$ of the softmax output and use it as the true tag for that sample. It is clear that in the first steps of the optimization process that tag would be far from the true tag and that means the adversarial noise will not push towards a the direction in which the cost is increase. This means adversarial noise will in someway resemble gaussian noise so there is no problem in adding it, because we are also adding gaussian noise. But as long as the network is well trained the adversarial noise will push the samples towards the decision bound. We can see this in figure 2.

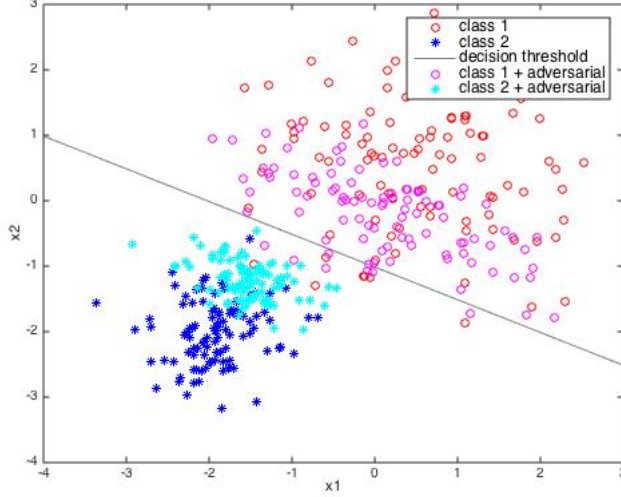


Figure 2: This figure represents two supervised data sets (dark red and dark blue) drawn from two 2D gaussian distributions and the decision bound. We have applied adversarial noise to both data set computing the light red and light blue datasets. As we can see both data sets are pushed towards the decision error and thus towards each other.

In this case we present a supervised data set to motivate our idea for unsupervised data. As we see samples correctly classified are pushed towards the decision threshold. We train adding unsupervised noise to force the network learn to denoise samples in sensible directions. This means extracting features to represent $p(x)$ bounding in someway how we want our network to represent that distribution, because we want to find features useful for supervised data.

On the other hand, bad classified samples are also pushed towards the decision threshold, and that means they are pushed towards the data set they belong to. This is an effect possibly present when adding gaussian noise.

4 Experiments

We perform preliminar experiments over MNIST. For this dataset we evaluated a fully connected network with 100, 1000 and all the labels and a convolutional network with 100 labelled data. Details on these networks topologies can be found in [Rasmus et al., 2015].

We add noise to the supervised data following the next expression:

$$\tilde{X} = X + \tau_a \cdot \text{sign}\left\{\frac{\partial C_s}{\partial x}\Big|_X\right\} \quad (8)$$

We add noise to the unsupervised data following:

$$\tilde{X} = X + \tau_b \cdot \frac{\partial C_s}{\partial x}\Big|_X; \left\|\frac{\partial C_s}{\partial x}\Big|_X\right\| = 1 \quad (9)$$

where τ_a and τ_b are hyperparameters searched for each task. For the unsupervised data we did not check to evaluate the sign function over the derivative because we think that the phase information of the derivative vector is important in learning $p(x)$. In the unsupervised case we restrict the norm vector to one to control the power of the noise. We could made the norm of the noise equal to the one computed with the sign function by setting $\tau_b = \sqrt{K}$ where K is the dimensionality of the input image. For the supervised data we tried both ways of computing the adversarial noise. Remark that these decisions were made over the Fully Connected Full Labelled MNIST task and evaluated in the other tasks.

In the next tables we show the results. We have also perform a experiment on the MNIST 100 label convolutional to show that adding adversarial noise to the unsupervised data is useful. The results are an average over 10 different runs.

FC MNIST			
# of labels	100	1000	All labels
DBM, Dropout [Srivastava et al., 2014]			0.79%
Adversarial [Goodfellow et al., 2014]			0.78%
Γ -model (Ladder with only top-level cost) [Rasmus et al., 2015]	3.06%	1.53%	0.78%
Ladder, only bottom-level cost [Rasmus et al., 2015]	1.09%	0.9%	0.59%
Ladder full [Rasmus et al., 2015]	1.108%	0.952%	0.565%
Adversarial Ladder full	1.084%	0.932%	0.560%

Table 1: Results for fully connected MNIST task. In boldface are state of the art results.

Convolutional MNIST	
# of labels	100
EmbedCNN [Weston et al., 2012]	7.75%
SWWAE [Zhao et al., 2015]	9.17%
Baseline: Conv-Small, supervised only [Rasmus et al., 2015]	6.43%
Conv-FC [Rasmus et al., 2015]	0.99%
Conv-Small, Γ -model [Rasmus et al., 2015]	0.89%
Adversarial (only supervised) Ladder Γ -model	0.80%
Adversarial Ladder Γ-model	0.79%

Table 2: Results for Convolutional MNIST task. In boldface is state of the art result.

5 Conclusions and Future Work

We see how we can improve the results of the ladder network by adding adversarial noise. Future work will be focus on four different lines. First one, removing the gaussian noise but adding adversarial noise computed for the unsupervised samples as we exposed. Second, adding supervised adversarial noise to each layer in the topology. Third, extending the validation with a wider experimentation with other tasks. Finally, depending on the results, mix the different explored approaches.

References

- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.
- I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- G. I. Hinton. In Y. Kodratoff and R. S. Michalski, editors, *Machine Learning*, chapter Connectionist Learning Procedures, pages 555–610. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 0-934613-09-5. URL <http://dl.acm.org/citation.cfm?id=120048.120068>.
- S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- Y. Lecun, F. Jie, and J. N. E. L. Com. Loss functions for discriminative training of energy-based models. In *In Proc. of the 10-th International Workshop on Artificial Intelligence and Statistics (AISTats05)*, 2005.
- Y. Lecun, S. Chopra, R. Hadsell, R. marc’aurelio, and f. Huang. A Tutorial on Energy-Based Learning. In G. Bakir, T. Hofman, B. schölkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2006.
- A. Rasmus, H. Valpola, M. Honkala, M. Berglund, and T. Raiko. Semi-supervised learning with ladder network. *CoRR*, abs/1507.02672, 2015. URL <http://arxiv.org/abs/1507.02672>.
- R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 1, page 3, 2009.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=2627435.2670313>.
- C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2014.
- H. Valpola. From neural pca to deep unsupervised learning. *Adv. in Independent Component Analysis and Learning Machines*, pages 143–171, 2015.

- P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning, ICML '08*, pages 1096–1103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294. URL <http://doi.acm.org/10.1145/1390156.1390294>.
- P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=1756006.1953039>.
- J. Weston, F. Ratle, H. Mobahi, and R. Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer Berlin Heidelberg, 2012.
- J. Zhao, M. Mathieu, R. Goroshin, and Y. Lecun. Stacked what-where autoencoders. *arXiv preprint arXiv:1506.02351*, 2015.